Graphical Tool For SC Automata.

Honours Project: 2000

Dr. Padmanabhan Krishnan[1] Luke Haslett

[1]Supervisor

Abstract

SC automata are a variation of timed automata which are closed under complementation. The major difference is SC automata have both history clocks which represent the time since some event occurred in the past and prophecy clocks which represent the time until some event occurs in the future. Humans have difficulty understanding and visualising the meaning of prophecy clocks and constraints which test their values.

A graphical tool for constructing SC automata and experimenting with their accepting runs is presented. The tools emphasis is to provide understanding and visualising prophecy clocks rather than being a solid verifier. A simple evaluation of the tool is also presented.

Contents

Chapter 1

Introduction

Timed automata are a common way to model real-time systems. As discussed in [AD94], timed automata are finite automata which are augmented with timing constraints. They can capture many interesting aspects such as liveness, fairness, periodicity, bounded response and timing delays. Unfortunately timed automata are not closed under complementation, resulting in the language inclusion between two timed automata to be undecidable. If they were closed under complementation, causing the language inclusion problem to become decidable, it would be theoretically possible to completely mechanise the verification process by taking an abstract description of a model and verifying it is the same as a more operational one. For this reason effort has been directed into finding variations of timed automata which are closed under complementation, but still remain expressively powerful enough to capture as many aspects of real-time systems as timed automata can.

State clock (SC) automata are an example of an automaton which are closed under complementation. Their main difference with timed automata's is it's set of clocks is broken into two sets, history and prophecy variables. As discussed in [RS97], SC automata are formally less expressive than timed automata but are still powerful enough to express the most general aspects of real-time systems, bounded response time, bounded invariance, timeout and others.

SC automata do have complications though. History variables record the amount of time elapsed since a proposition was previously true and are easy to comprehend, as are constraints which test the values of these clocks. Prophecy variables on the other hand are difficult to comprehend because their values represent the time until a proposition will be true in the future, and constraints can test the values of the clocks before their respective propositions occurs. Intuitively to have values for prophecy clocks and have constraints test their values, the path followed in an automaton will already be known so that the time until these events will occur can be calculated. From a human perspective, as we follow an accepting run of an SC automaton from start until finish, the meaning of the current value of a history variable makes sense because we can remember the path we have taken so far and can probably remember when a clocks associated propositions were last true. The prophecy variables are not so simple, we don't know what future path we will take, so the current value of a prophecy variable has little meaning to us at this point of time until we know where its proposition will next become true. For the same reason constraint

involving prophecy variables will also have little meaning to us.

1.1 Tool objectives

As SC automata are closed under complementation, resulting in the language inclusion problem to be decidable, a graphical tool which determines the acceptance of a runs would be the first step towards creating an automatic verifier for SC automata models. The primary objective of the tool though is to aid in the visualising of constraints involving prophecy variables and providing more information on how these prophecy variables obtained their current values. The emphasis is on why a run is accepted rather than if a run is accepted.

1.2 Report Structure

Chapter 2 introduces both timed automata and SC automata. The definition of each is presented along with examples.

In Chapter 3 the design of the new tool is described. It discusses briefly some other tools for different automata then on describing the actions users will want to do when constructing an automaton and experimenting with an accepting run.

In Chapter 4 the implementation of the tool is discussed. Using an example the steps taken to create an automata and then experimenting with its accepting run is shown.

In Chapter 5 an evaluation of the tool is presented. It focuses on what its good and bad points are and what aspects weren't implemented and should have been.

Chapter 6 presents the conclusions and ideas for future development.

Acknowledgements

I would like to thank Dr. Padmanabhan Krishnan for his assistance with this research and Jane McKenzie for proof reading this paper.

Chapter 2

Background

2.1 Timed Automata

Timed automata are finite automata which are augmented with a set of clocks and timing constraints [AD94]. There are a finite number of these real-valued clocks which keep track of the time elapsed since each clock was last reset. Associated with each transition is a symbol, a set of clocks to reset, and a constraint. The constraint must be satisfied (evaluate to true) by the current value of these clocks in order to take the associated transition. A constraint is defined by:

$$\delta \quad := \quad w \sim c \mid w_1 \sim w_2 \mid \neg\delta \mid \delta_1 \wedge \delta_2$$

where w, w_1, w_2 are clocks, c is a positive real-valued constant, $\delta, \delta_1, \delta_2$ are clock constraints and $\sim \in \{<, \leq, =, \geq, >\}$.

Timed automata accept timed words - an infinite sequence in which a real-valued time of occurrence is associated with each symbol.

Example 1

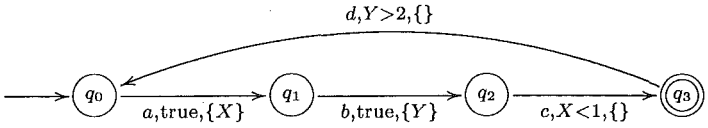

Figure 2.1: Example Timed Automaton

Figure 2.1 is an example timed automaton. This automaton accepts timed words with the repeated sequence of symbols 'abcd'. It has two clocks associated with it. The clock X is reset when the symbol 'a' is read (the transition between states q0 and q1). When the symbol 'c' is read the constraint $X < 1$ must be satisfied. This implies that the time between an 'a' and a 'c' must be less than one time unit. Similarly greater than two time units must pass between reading a 'b' and reading a 'd'.

2.2 SC Automata

2.2.1 Definition 1: SC automata

SC automata [RS97] are quite different to timed automata. They are state based rather than transition based. Each location (state or node) is labelled with propositional symbols and constraints. There is a finite set of propositions associated with an automaton and each location is labelled with a set of propositions which are true at this location. Each proposition has a history and prophecy variable, denoted x_p and y_p respectively, which makes up the set of clocks. The value of the history variable represents the time elapsed since the proposition was last true. The value of the prophecy variable represents the time until the proposition will next be true. Constraints are similar to those of timed automata except that clock's values can not be compared with each other and constraints must remain satisfied while at the associated location. Constraints have the form:

$$\delta \quad := \quad w \sim c \mid \neg\delta \mid \delta_1 \wedge \delta_2$$

where w is a clock's, c is a positive real-valued constant, $\delta, \delta_1, \delta_2$ are clock constraints and $\sim \in \{<, \leq, =, \geq, >\}$.

Example 1

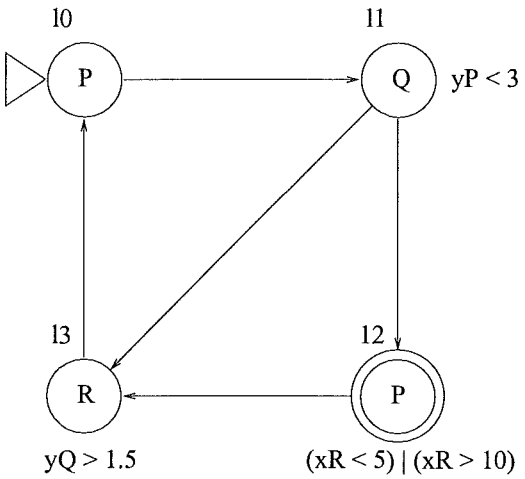

Figure 2.2: Example SC Automaton

Figure 2.2 is an example SC automaton. It has a set of four locations $\{l_0, l_1, l_2, l_3\}$, with start location l_0 and final location l_2. It has the set of propositions $\{P, Q, R\}$. Each of these propositions is either true or false in each location. At location l_0 P is true, and Q and R are both false.

The constraint $(xR < 5)||(xR > 10)$ at location l_2 demonstrates the use of xR, a history variable over proposition R. It states that while at location l_2 ,

the time elapsed since R was last true must either be less than 5 time units or greater than 10 time units. The constraint $yQ > 1.5$ at location l_3 demonstrates the use of yQ, a prophecy variable over proposition Q. It states that while at location l_3 the time until Q will next be true must be greater than two time units.

2.2.2 Definition 2: A timed sequence of states

A timed sequence of states is represented as a sequence of couples. A couple contains a time interval and a set of propositions which are true during this time interval. For example the couple $(\{P\}, [0, 1.5])$ states that from time 0 until time 1.5 the proposition P is true. The left end point of the time interval is 0, and the right end point is 1.5. Each time interval must be adjacent to its neighbours, forming a sequence of intervals. Right end points must be equal to the next time interval's left end point.

Example 2

Let us consider the following timed sequence of states for the SC automata defined in Example 1 (Figure 2.2):

$$m = (P,[0, 1.5]) \ , \ (Q,[1.5, 4]) \ , \ (R,[4, 4.3]) \ , \ (P,[4.3, 6]) \ , \ \ldots$$

In the first state of the sequence the proposition P is always true, and Q and R are always false. We remain in this state until time 1.5, when we move to the second state where Q is always true and P and Q are always false.

Each interval is adjacent to the next. For instance in the first state of the sequence the right end point is 1.5, as is the left end point of the second state.

For timed sequences of states, intervals will always be left closed, right closed. [RS97] tells us that physically it is impossible to determine if an interval is closed or open. As it is invariant to the form used we will always use the form left closed, right closed. This simplifies the construction of timed words and implies that at an end point of an interval we will be in two states. Using the example above, at time 1.5 we are in both the first state and the second state.

2.2.3 Definition 3: Clock valuation

The value of the history and prophecy variables along a timed sequence of states can be determined by:

- **For the history variable:**
  - While in a state where the proposition is true, the value of its history variable is 0.
  - While in a state where the proposition is false, the value of its history variable is the time elapsed since the proposition was last true.
  - While in a state where the proposition is false, if the proposition has never been true so far then the value of its history variable is undefined.
- **For the prophecy variable:**

- While in a state where the proposition is true, the value of its prophecy variable is 0.
- While in a state where the proposition is false, the value of its prophecy variable is the time the proposition will next be true.
- While in a state where the proposition is false, if the proposition will never be true again then the value of its prophecy variable is undefined.

Example 3

Using the SC automaton defined in Example 1 (Figure 2.2) and the timed sequence of states m defined in example 2, primarily focusing on the second couple (Q, $[1.5, 4)$) at time $t = 2.4$, intuitive meanings of the two definitions above are as follows:

- **History Variable**:
  - At time $t = 2.4$, the value of the history variable $x_Q = 0$ because the proposition Q is true from time 1.5 until time 4.
  - At time $t = 2.4$, the value of the history variable $x_P = 0.9$. The last time that proposition P was true was at time $t = 1.5$ when leaving the first state of the sequence m.
  - At time $t = 2.4$, the value of the history variable x_R is undefined, because the proposition R has never been true.
- **Prophecy Variable**:
  - At time $t = 2.4$, the value of the prophecy variable $y_Q = 0$ because the proposition Q is true from time 1.5 until time 4.
  - At time $t = 2.4$, the value of the prophecy variable $y_P = 1.9$. The next time that proposition P will be true is at time $t = 4.3$ when entering the forth state of the sequence m.
  - At time $t = 2.4$, the value of all prophecy variables are defined because all of them are true at some point in the future.

2.2.4 Definition 4: An accepting run of an SC automaton

An accepting run of an SC automaton on a timed sequence of states is a sequence of couples. Each couple contains a location and a time interval. During each time interval the run is at the associated location. The run is accepted if the following conditions are satisfied:

- **Initially**: The location in the first couple is the start location l_0.
- **Interval**: The intervals of the timed sequence of states forms an interval, as discussed in section 2.2.2.
- **Adequation**: The propositions that label the locations visited at time t is equal to the set of propositions that are true at time t in the timed sequence of states.

- **Consecution**: Each adjacent location is either the same location or there is a transition between them.
- **Timing constraints**: The constraint associated with the current location remains true over the entire time interval.
- **Acceptance**: In the last couple of the run, the associated location is a final location in the automaton.

Example 4

Let us consider the following accepting run over the sequence of states m we defined in Example 2 for the SC automaton defined in Example 1 (Figure 2.2).

$$r = (l_0, [0, 1.5]), (l_1, [1.5, 4]), (l_3, [4, 4.1]), (l_3, [4.1, 4.3]), (l_1, [4.3, 6]), \ldots$$

The run r is accepted because each constraint is satisfied. The following is an intuitive explanation of how each constraint can be satisfied:

- **Initially**: The location l_0 is the location in the first state of the run r.
- **Interval**: As explained for example 2, each interval is adjacent to the next. For instance in the first state of the run r the right end point is 1.5, which is the same as the left end point of the second state. This is true for every state in the run r.
- **Adequation**: From time 0 until time 1.5 we remain in state l_0 of the SC automaton where proposition P is true. Over the same time period we remain in the first state of the sequence m with proposition P true. This is true for all other time intervals.
- **Consecution**: As an example of staying in the same state, the third and forth state of the run r both refer to the same location l_3. Intuitively we have just broken one possible state $(l_3, [4, 4.3])$ into two distinct states $(l_3, [4, 4.1])$ and $(l_3, [4.1, 4.3])$.
- **Timing constraints**: The constraint at a location must be satisfied for the entire duration of time we spend there. Imagine the sequence m continues as follows:

  $$m = \ldots, (R, [4, 4.3]), (P, [4.3, 6]), (Q, [6, 7]), (P, [7, 9]), \ldots$$

  At time $t = 7$ we enter location l_2 of the SC automaton and must satisfy the constraint $(x_R < 5) \| (x_R > 10)$. This constraint must be satisfied at every point in time until time $t = 9$ when we leave location l_2. At time $t = 7$ the value of the history variable x_R is 2.7. At time $t = 9$ the value of the history variable x_R is 4.7. For every point in time between $t = 7$ and $t = 9$, the value of $x_R < 5$ so the constraint is satisfied.

  Let us now say we have the following timed sequence of states m' with one small change to the final state of the sequence m:

  $$m' = \ldots, (R, [4, 4.3]), (P, [4.3, 6]), (Q, [6, 7]), (P, [7, 17]), \ldots$$

At time $t = 7$ the value of the history variable x_R is 2.7. The value of $x_R < 5$ so the constraint is satisfied at this time. At time $t = 17$ the value of the history variable x_R is 12.7. The value of $x_R > 10$ so the constraint is satisfied at this time. From time $t = 9.3$ until time $t = 14.3$ the constraint is not satisfied because $x_R \not< 5$ or $x_R \not> 10$. The run m' is therefore not accepted.

- **Acceptance**: Once at the last state for the timed sequence of states, the location associated with the interval for this state will be a final location.

Chapter 3

Design

As SC automata are closed under complementation, resulting in the language inclusion problem to be decidable, a tool which determines the acceptance of a runs would be the first step towards creating an automatic verifier for SC automata models. The primary objective of the tool though is to aid in the visualising of constraints involving prophecy variables and providing more information on how these prophecy variables obtained their current values. The emphasis is on why a run is accepted rather than if a run is accepted.

The steps taken to design the new tool was firstly to have a look at a couple of other tools for ideas and secondly describe the actions a user should go through when constructing an automaton and experimenting with runs it accepts.

3.1 Other tools

3.1.1 JFLAP

JFLAP [RPP98] is a package containing a number of graphical tools for use in learning the basics of automata theory. One of its tools is a finite state automata (FA) experimenter. It allows the user to graphically construct an FA and test if it accepts or rejects an input string. It also allows you to determine its non-deterministic states and convert it to a deterministic FA.

3.1.2 UPPAAL

UPPAAL [YL00] is a very complete timed automata package. It allows you to construct, simulate and verify. Its focus is more on debugging models, running them concurrently with others and verifying properties rather than on providing understanding on the values of clocks and their constraints. It provided some hints on separating the interface up into distinct construction and experiment areas. The new SC automata tool has different objectives to what UPPAAL does.

3.2 User Actions

The focus of the new tool is on providing understanding of prophecy variables and its associated constraints primarily, having an effective and efficient analyser and verifier is secondary. It is therefore most important to focus on actions the user will do and in what order they will do it.

Their are two distinct activities that the user will do, construct an automaton and experiment with the automatons accepting runs.

3.2.1 Construction

Figure 3.1 shows the steps a user will take to construct an automaton.

The first step a user will do is decide on their set of propositions, which will also define the sets of history and prophecy variables.

The second step will be to either construct the actual automaton (with locations, transitions and constraints) or to construct the timed sequence of states. Order is not important at step two but in practice it is expected that a user would firstly construct an automaton then secondly construct and experiment with differing timed sequences of states.

When constructing the actual automaton the locations will have to be constructed first. Once they are constructed then the location can be configured to be initial, final, or the propositions that are true at this location are set. Transitions can then be constructed between two locations and constraints can be constructed and associated with a location.

Some of these activities had interesting design decisions and are discussed in greater detail below.

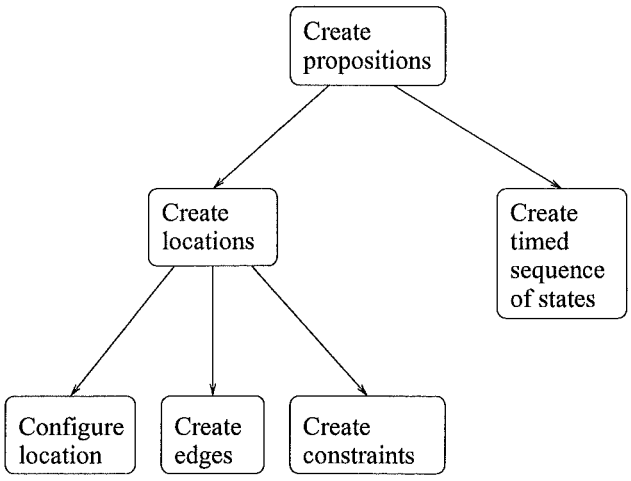

Figure 3.1: Order of steps to construct an automaton

Actual automaton construction

SC automata have timing constraints, propositions and are state rather than transition based. At the lowest level though they still have locations, transitions, final and initial locations, so initially following the same steps as you would follow to construction an FA using JFLAP is suitable. Users of JFLAP should feel accustomed to the steps required to start building an SC automata when using the new tool.

Constraint construction

As defined in section 2.2.1 a constraint can have the form:

$$\delta \quad := \quad w \sim c \mid \neg\delta \mid \delta_1 \wedge \delta_2$$

δ will be referred to as a constraint component. There are two distinct stages when constructing a constraint.

The first is at its simplest level a constraint component is $w \sim c$, made up of a clock variable, relational operator and a real valued constant. The facility to construct these simple constraint component easily needs to exist.

The second is when constraint components can then be combined using the rules $\neg\delta$ and $\delta_1 \wedge \delta_2$. As complementation and intersection are both possible, it is also possible to perform union, equivalence and implies between two constraint components. The facility to combine constraint component also needs to exist.

Timed sequence of states construction

As defined in section 2.2.2 a timed sequence of states is a sequence of couples. Each couple contains a set of propositions which are true and a time interval. Each time interval must be adjacent to its neighbours.

Users will construct one couple at a time appending each new couple to the sequence of already constructed couples. To ensure that each successive couple is adjacent to the next the right end point for one couple will become the left end point for the next automatically.

3.2.2 Testing Accepting Runs

There are two stages to this activity, first testing if a run accepts and secondly experimenting with it.

Determining acceptance of a run

Essentially an accepting run is a timed sequence of states which is accepted by an automaton. To accept, at every moment in time, the propositions which are true in the current couple must be the same as the propositions which are true at the current location and the current values of the clocks must satisfy the locations associated constraint.

The algorithm used to determine acceptance is simple, it consists of two steps:

1. Beginning at the start location and in the first couple of the timed sequence of states traverse from one couple to the next and one location to the next

making sure that the propositions true for both the couple and the location are the same. What propositions are true in the current couple dictate what location we can be in. Continue this until the end of the timed sequence of states is reached and the location is a final state. Ignore constraints for the moment but record the value of the history variable upon entering each location. If no path was found that finished at the end of the timed sequence of states and at a final location then the run is rejected, otherwise continue to the next step.

2. Beginning at the final location and in the final couple of the timed sequence of states traverse backwards following the reverse path determined from the previous step until back at the start location and in the first couple. Set the value of the prophecy variables at each step because we now know when each proposition will be true in the future. The constraints can know be checked because values exist for all history and prophecy variables. Constraints must be true for the entire duration remained at a location so all points where constraint components change from true to false or vice versa must be checked. If each constraint has been satisfied and the run is at the start location and in the first couple then the run is accepted, otherwise it is rejected.

The first step is trivial and is essentially what is used in JFLAP. At this point the algorithm can handle non-deterministic SC automata, it traverses the search space in a depth-first search manner until it finds a legal path to an acceptance state or it exhausts all possible paths.

The second step is still relatively trivial but there are some catches. As prophecy variables rely on future events the simplest way is to fill their values in on the way back to the beginning from the end once we know the path taken. Unfortunately if a constraint fails then the over all run is rejected, an alternative path is not tried so the algorithm only works on deterministic SC automata. But it is possible to convert any non-deterministic SC automaton into a deterministic SC automaton [RS97]. As long as the user ensures that the SC automaton is deterministic (which is always possible), then the algorithm will work correctly.

Experimenting with an accepted run

This design follows JFLAP technique at its simplest level. Once a run is accepted you then get to experiment with the accepted run. Initially you start in the start location and the first couple in the sequence of states. You have the freedom to then go back and forth through the path of the accepting run. While experimenting with an accepting run the following information will be available:

- The value of any history of prophecy variable at any point in time.
- The simple constraint components which are true and which are false for the constraint of the current location to be satisfied at any point in time.
- It should be easy to understand how a constraint is true for an entire time interval.

- From a given point in time the path taken to where a proposition was last true or is next true is viewable.

Chapter 4

VSC: The program

The following sections describe how VSC [Has00] the implemented visual SC automata tool works . Figure 4.1 shows the main window which is permanently on display while using the tool. The actual automaton is constructed on the left side of this display. The timed sequence of states is permanently displayed at the top and propositions are permanently displayed on the right side of the display. Is you want to test a run, add propositions or change the timed sequence of states this is all done with the menus at the very top of the display.

Example

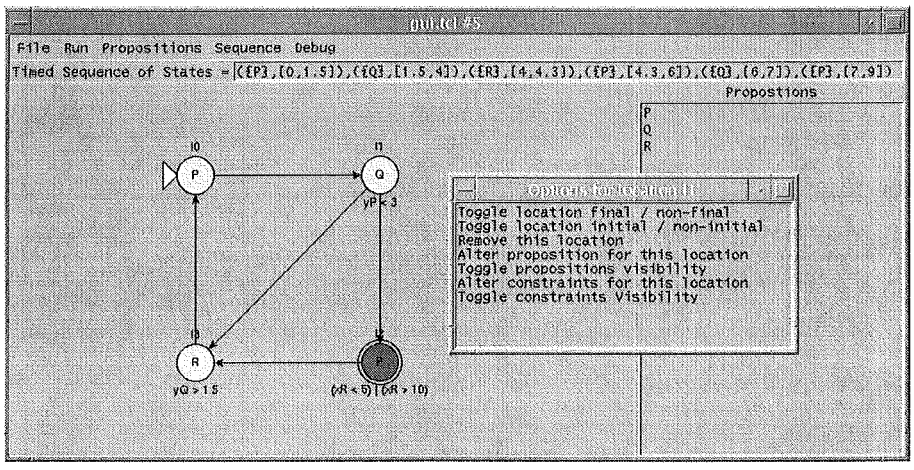

Figure 4.1: Main window

Figure 4.1 shows the main window displaying the automaton from Examples 1-4 in Chapter 2.

Propositions are created through the *Propositions* menu. In our example the set of propositions $\{P, Q, R\}$ are permanently displayed on the right side of the display .

Locations and transitions are created simply by clicking on the left side of the display. Each location has its name above it ($l0, l1, l2, l3$), its true propositions at its center and its constraint listed below. Right clicking on a location brings up an *options* menu where locations can be made initial of final, each proposition can be made true or false, or a constraint can be created.

4.1 Construction of constraints

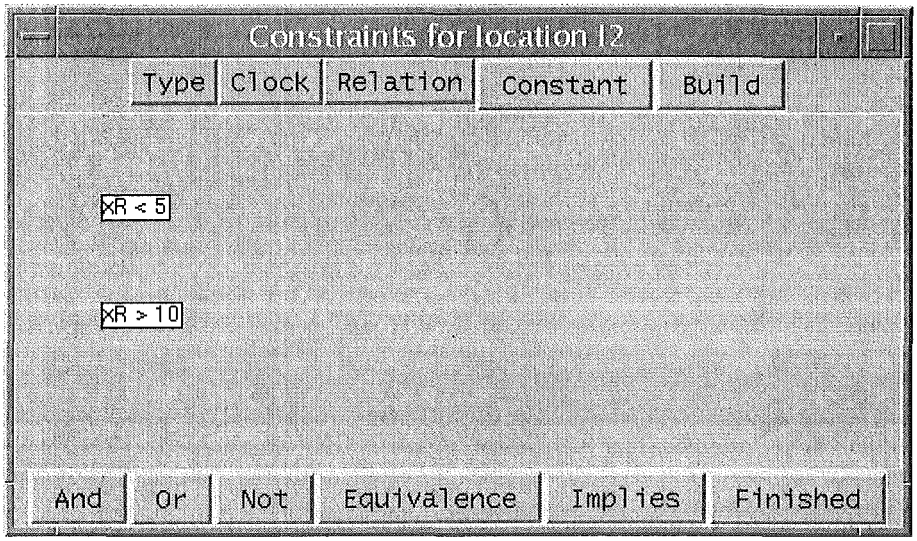

Figure 4.2: Build a Constraint

Figure 4.2 shows how a new constraint is constructed. This window is brought up by a selection in the options menu for a location. It takes two stages to construct a constraint.

The first stage is to construct the simple constraint components. A simple constraint component has the form of a clock variable, then some relational operator followed by a real valued constant (clock $\sim$ constant, where $\sim$ is in $\{<, \leq, =, \geq, >\}$). The buttons at the top are used to do this. The *Type* button allows you to select whether a variable is a history or prophecy variable, *clock* allows you so select the proposition the clock is associated with, *Relation* specifies the relational operator, *Constant* allows you to fill in the real valued constant, and *Build* creates the constraint component from the information provided and places it in the middle section of the display. From the example, two simple constraint components where constructed, $x\mathrm{R} < 5$ and $x\mathrm{R} > 10$ and once built are displayed in the middle section of the display.

The second stage is to combine the constraint components into the overall constraint. This is done with the buttons at the bottom. In our example if we click on the two constraint components $x\mathrm{R} < 5$ and $x\mathrm{R} > 10$ and then on the *Or* button, a new constraint component $(x\mathrm{R} < 5)|(x\mathrm{R} > 10)$ will be constructed. Each of these buttons require two constraint components to be selected except *Not* which requires one constraint component and compliments it, and *Finished*

which requires one constraint component and exits this window, labelling the associated location with the new constraint.

4.2 Construction of timed sequence of states

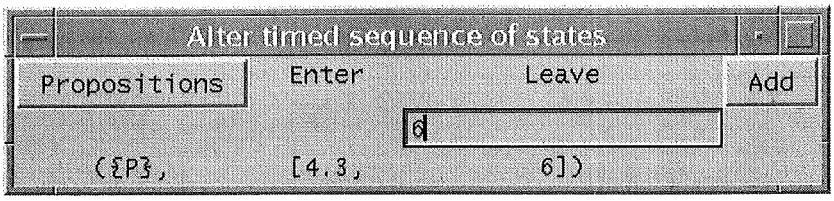

Figure 4.3: Building a times sequence of states

Figure 4.3 shows how the couples of a timed sequence of states are created. The *Propositions* menu allows the selection of which propositions will be true for the duration of the new time interval. The *Enter* column displays the leave value of the previous couples time interval or zero if this is the first couple in the sequence, which insùres that the intervals are adjacent and form a sequence. The *Leave* column allows the insertion of a real-valued constant beneath it and this forms the right end point of the time interval and the enter value of the next couples time interval. The *add* button appends the new couple to the already existing sequence of states, the enter value changes to the current leave value, and the next couple can be constructed.

Using the example discussed previously Figure 4.2 shows the construction of the forth couple in the sequence. When add is selected this new couple will be displayed on the main window and appended to the end of the couples already created, the value of the left end point in the enter column will change to 6, and the construction of the fifth couple can begin.

4.3 Testing accepting run

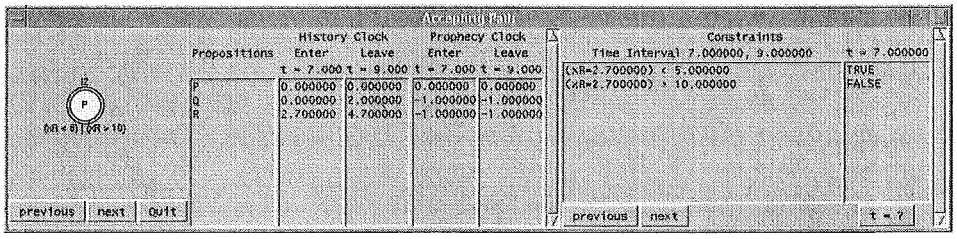

Figure 4.4: Testing an accepting run

Figure 4.4 shows the window that is displayed when a run is accepted and we want to experiment with it. Initially we begin in the start location ($l0$) and in the first couple of the time sequence of states.

The left side of the window shows the location we are currently at. We are currently at location $l2$ and in the sixth couple of the timed sequence of states. Selecting *next* will take us to the seventh couple and a new location. Similarly selection *previous* will take us to the fifth couple and a new location. The current location shown here on the left is highlighted red on the main window to give an indication of where we are. This is the primary way to navigate through a run and back again. It shows us what the accepting path was for the run.

The center of the window shows the values of the history and prophecy variables. The propositions are listed down the far left column. The second and third columns represent the values the history variables have on entering the new location and the values they have when leaving. The forth and fifth columns represent the value the prophecy variables have on entering the new location and the values they have when leaving. For example the history variable for proposition R (xR) has the value 2.7 when entering location $l2$ at time $t = 7$ and will have value 4.7 when leaving location $l2$ at time $t = 9$. Values of -1 indicate an undefined value.

The right side of the window is to help explain why the constraint is true while at this location. Each simple component of the constraint is shown on a separate line on the left. On the right a time is displayed at the top and then it shows which constraints are true and which are false at this point in time. At time $t = 7$ the constraint $x\mathrm{R} < 5$ is true and $x\mathrm{R} > 10$ is false. The *next* button takes you to the next point in time > 7 that one of the constraints changes from either true to false or vise-versa. This is to show that at every point in time during this time interval that the overall constraint remains true. Selecting the button at this moment will take us to time $t = 9$ because no simple constraint component changes so we go all the way to the end of the time interval. The $t =?$ button takes us to any point in time in the time interval and shows what simple component constraints are true and which are false. We could select to go to time $t = 8$ and see which components are true and which are false. Also we can select a constraint component and a blue path will be shown on the main window, showing what the path is to where the associated propositions is true. If we were to select either of the two constraint component in the example, then the path to where Proposition R was last true would be $l3 \rightarrow l0 \rightarrow l1 \rightarrow l2$ (see Figure 4.5). Each of these location is displayed in blue and the transitions $l3 \rightarrow l0$, $l0 \rightarrow l1$ and $l1 \rightarrow l2$ are also displayed in blue.

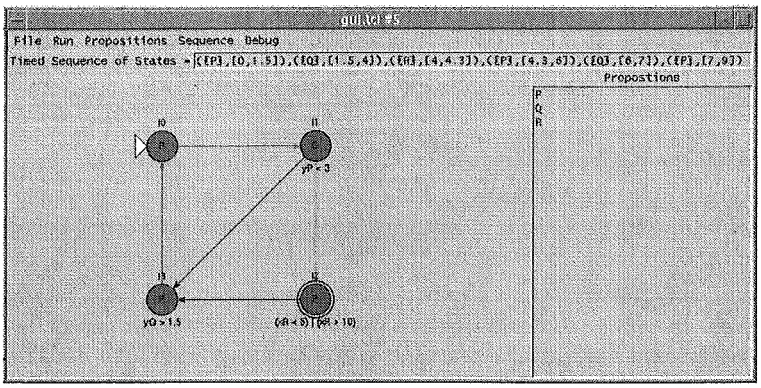

Figure 4.5: Path showing location where proposition R was last true

Chapter 5

Evaluation

In most area the tool can be considered a success. The construction and experimenting with accepting runs is adequate and the understanding of the prophecy variables and there constraints is very useful. A summary of significant advantages and disadvantages for different parts of the tool is discussed below.

5.1 Construction of locations, transitions, propositions

Advantages

- Very similar to the methods used in JFLAP. Those familiar to JFLAP would be very familiar with how to construct locations and transitions.
- Propositions are permanently displayed on the right side of the main window allowing an easy way to view what propositions are available. From the set of propositions which were defined the tool automatically defines the history and prophecy variables.

Disadvantages

- A few little things such as deletion of location, transitions and propositions were not implemented. The movement of locations was also only partially completed.

5.2 Construction of constraints

Advantages

- There was a good separation between constructing a simple constraint component and then combining them into one single constraint using boolean operations.
- Constraints were not typed out manually, they were constructed graphically. This made it faster to construct and less error prone.

Disadvantages

- Blanking out options that weren't available wasn't implemented. For instance complementation required one constraint component but all the others required two, it isn't always clear what you options are.
- There was no way to alter a constraint once it was finished.

5.3 Construction of timed sequence of states

Advantages

- The tool automatically created sequence of adjacent couples be making the right end point of the previous couple the left end point of the current couple.
- Each couple was simply appended to the and of the currently created sequence.

Disadvantages

- There was no way to modify a sequence once it was finished, only to clear it and start again. Would have helped when experimenting.

5.4 Experimenting with accepting runs

Advantages

- The general method of stepping back and forth through a run is the same as in JFLAP.
- Although it was not possible to see the values of a history or prophecy variable at any point in time, the tool did display the values of all clock values for entering and leaving the current location, you could determine a specific value your self if need be.
- The right side of the display showed very effectively why a constraint was true while at a location. It was possible to check any point in time and see what constraints were true or just be taken on a tour of important instances of time during the time interval when simple constraint components change.
- The blue path displayed on the main window was adequate to show where the proposition was true.

Disadvantages

- No indication of why a rejected run was rejected.
- No visible representation of where in the timed sequence of states you currently are.

- If the blue path on the main window went to the same node twice then the path would have been confusing. If a delay had been placed between drawing each location and transition you could have followed step by step what the path was as it was drawn.
- Although it was possible to check each simple constraint component and see which are true or false there was no support offered for once the constraints are combined in any way, other than knowing the overall result is always true. One method proposed which would aid this is to represent the entire constraint as a boolean circuit, representing each boolean operation as a gate and use different colours to show which are true and which are false.

Chapter 6

Conclusion

As a construction tool it works well. It follows JFLAP construction methods closely. The construction of both constraints and timed sequences of states are simple and avoid errors. Its problems in this area is modification and deletion but these are only trivial and are easy to implement.

When experimenting with an accepting run a high level of understanding is provided to a user as to why variables have the values they do and where their propositions are true in the acceptance path.

Users are able to easily experiment and determine why constraints are true during an entire time interval. Its failing in this area is that as it is easy to see why simple constraint components are true there is no current facility to determine which parts are true once simple constraint components are combined using boolean operations. A boolean circuit is proposed to solve this but is not implemented.

6.1 Future work

A great deal could be done with the tool from here. The boolean circuit proposed to add more meaning to constraint evaluation could easily be implemented. Other feature could be added such as loading and saving to file, modification of constraints and timed sequence of states to improve users productivity.

There are several non-trivial additions which could also be made. As the acceptance algorithm does not work on non-deterministic automaton a facility to convert from non-deterministic to deterministic automaton could be added. As SC automata are closed under complementation and the language inclusion problem is decidable a fully automatic verifier could be constructed. At present there is no assistance for runs which are rejected, being able to experiment with paths that failed and see where constraints failed would prove interesting.

Bibliography

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, (126):183–235, 1994.

[Has00] L Haslett. Vsc, computer program. *http://www.cosc.canterbury.ac.nz/ lah41/VSC/*, 2000.

[RPP98] S Rodger, M Procopiuc, and O Procopiuc. Jflap, computer program. *http://www.cs.duke.edu/ rodger/tools/tools.html*, 1998.

[RS97] J-F. Raskin and P-Y. Schobbens. State clock logic: A decidable real-time logic. *Hybrid and Real-Time Systems*, (1201):33–47, 1997.

[YL00] W Yi and K Larsen. Uppaal, computer program. *http://www.uppaal.com*, 2000.